PIF1006 – Mathématiques pour informaticiens II

Session Automne 2021

TRAVAIL #2

Indications principales	
Date de présentation de l'énoncé	11 novembre 2021
Date de remise du travail	10 décembre 2021
Politique sur remises tardives	-25% par jour de retard
Éléments et format de remise	Rapport et fichiers complet de la solution ou du projet remis sous forme électronique sur le portail de cours dans un seul fichier compressé (.zip)
Pondération	10% (Partie #1 : 5%; Partie #2 : 5%)
Nb d'étudiants par équipe	2 à 5

INTRODUCTION

Le monde de l'informatique fondamentale ou appliquée ne peut se passer du concept de matrice. Nous retrouvons l'utilisation des matrices dans la plupart des branches connexes à l'informatique, comme les mathématiques, l'économie, les statistiques, l'intelligence artificielle, la réseautique, ou encore la physique, la chimie, et plus encore. C'est donc dire que ce concept est incontournable. Par ailleurs, la manipulation des matrices dans des programmes informatiques demeure légèrement plus compliquée qu'on ne se l'imagine. Cela engage très souvent des connaissances poussées, mais aussi un sens de la programmation bien aiguisé.

Un cas d'utilisation des matrices est la résolution d'un système d'équations. Ce cas est d'ailleurs le sujet du présent travail, qui s'organisera en deux parties. Nous avons vu en classe qu'il existait des méthodes classiques qu'on pourrait qualifier de « directes » qui permettent d'arriver à une solution « exacte » (pour autant que le déterminant ne soit pas nul), mais pour de grandes matrices ces méthodes ont l'inconvénient de demander beaucoup de temps de calculs à cause notamment de l'aspect récursif inhérent au calcul des déterminants. Pour contourner ce type d'handicap, qui pourrait être majeur si on n'y prête pas attention, nous avons aussi vu que l'analyse numérique offre des solutions approximatives (selon un taux d'erreur admis), mais tout à fait satisfaisantes. Une méthode de calcul inspirée de l'analyse numérique en l'occurrence la méthode de Jacobi offre justement une solution élégante et facile à implanter, quoiqu'elle ne permette pas toujours de converger vers une solution.

- Dans une première partie, vous serez amené tout d'abord à programmer une matrice et certaines propriétés ou opérations matricielles, sans avoir recours aux librairies prédéfinies dans le langage que vous choisirez. Cette classe de matrice que vous aurez alors programmée sera utilisée dans la seconde partie.

- Ainsi, dans un deuxième temps, vous devrez « *algorithmiser* » **deux des trois méthodes** classique directe de résolution d'un système d'équations. Vous ne vous préoccuperez que des cas où le déterminant est non nul. Vous devrez également implémenter une méthode itérative inspirée du calcul numérique pour la résolution de tels systèmes, soit la méthode de **Jacobi**. Vous devrez enfin donner un cas pour lequel cette méthode ne fonctionne pas et expliquer pourquoi la solution ne converge pas dans ce cas.

La suite de l'énoncé vous expliquera les détails et exigences pour chacune des parties.

PARTIE #1 : MATRICES (5%)

Dans cette première partie du travail #1, vous devez implémenter une matrice et certaines propriétés et opérations sur des matrices (sans évidemment avoir recourt à des classes de librairies déjà prédéfinies pour les matrices). La classe « ***Matrice »*** que vous créerez sera utilisée pour la partie #2 du travail qui sera expliquée par la suite.

Plus précisément, au minimum, vous devez créer une classe « ***Matrice*** » pour laquelle chaque instance :

- Possède la variable d'instance suivante :
    - ***double[,] matrice*** : tableau à 2 dimensions de nombre décimaux, qui stocke la matrice.
- Permet les opérations suivantes :
    - ***Matrice Additionner(Matrice)*** : l'addition d'une matrice, qui retourne une matrice;
    - ***Matrice FaireProduitScalaire(double)*** : Le produit de la matrice par un scalaire, qui retourne une matrice ;
    - ***Matrice FaireProduitMatriciel(Matrice)*** : Le produit matriciel (avec une autre Matrice), qui retourne une matrice.
        - **Vous devez prévoir une version de cette méthode qui prend en paramètre un certain nombre de matrice à multiplier et qui permet de donner également le nombre d'opérations de produits effectués pour le calcul du produit matriciel.**

- ***Bool EstTriangulaire(…)***: retourne vrai ou faux selon si la matrice est triangulaire ou non ; comme arguments de la méthode, un premier paramètre doit dire si on souhaite vérifier si la méthode est triangulaire inférieure, supérieure ou peu importe, et un second paramètre doit indiquer soit on souhaite vérifier si elle est triangulaire stricte ou non.

- Possède les propriétés suivantes <u>en lecture seule</u> (en Java, utilisez des méthodes du style « ***GetNomPropriété()*** ») :

  - ***double Trace*** : calcule et retourne la trace de la matrice, si la matrice est carrée ;

  - ***double Determinant*** : calcule et retourne le déterminant de la matrice, si la matrice est carrée ;

  - ***Matrice Transposee*** : calcule et retourne la transposée de la matrice ;

  - ***Matrice CoMatrice*** : calcule et retourne la comatrice de la matrice, si la matrice est carrée ;

  - ***Matrice MatriceInverse*** : calcule et retourne la matrice inverse, si la matrice est carrée ;

    - Avant de tout calculer, assurez-vous que la matrice est régulière;

    - Si la matrice est, en plus, triangulaire, calculez directement la matrice inverse en utilisant le schème

$$A^{-1} = \begin{vmatrix} \frac{1}{a_{1,1}} & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{2,2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{a_{n,n}} \end{vmatrix}$$

  - ***Bool EstCarree*** : retourne vrai ou faux selon si la matrice est carrée ou non;

  - ***Bool EstReguliere*** : retourne vrai si la matrice est régulière (déterminant non nul) ou faux si la matrice est singulière (déterminant nul).

Pour l'accès et l'édition de la matrice, vous pouvez également ajouter une façon de pouvoir accéder ou modifier un élément quelconque de la matrice d'une façon de votre choix, par exemple :

- ***En utilisant une propriété pour accéder au tableau contenant les éléments, comme :***

  - ***double[,] Elements***

- ***En utilisant des méthodes permettant d'obtenir ou de modifier un élément particulier du tableau, comme*** :
    - ***double GetElement(int ligne, int colonne)***
    - ***void SetElement(int ligne, int colonne, double valeur)***

Ajoutez aussi une méthode permettant d'afficher la matrice à la console ou dans un contrôle utilisateur (p. ex. en redéfinissant la méthode *ToString()* ou une méthode similaire).

- Il est évidemment à prévoir que l'implémentation de certaines opérations exige de définir des méthodes privées supplémentaires.
- Enfin, assurez-vous du bon fonctionnement de chacune des méthodes, par exemple en vous servant des exercices et leurs corrigés et/ou les exemples dans les notes de cours.

PARTIE #2 : SYSTÈMES DE N ÉQUATIONS LINÉAIRES À N INCONNUES (5%)

Dans cette deuxième partie, vous devrez implémenter une classe représentant un système de n équations linéaires à n inconnues et implémenter certaines des méthodes de résolution vues en théorie qui permettent de trouver les valeurs des inconnues.

Vous devrez implémenter deux méthodes donnant des solutions directes et exactes, ainsi que la méthode itérative de Jacobi pour la résolution de tels systèmes d'équations.

Pour ce faire, vous devez créer une classe « ***Système »*** pour laquelle chaque instance, minimalement :

- Possède les variables d'instance suivantes dont vous donnerez accès en utilisant des propriétés (ou des fonctions *GetXXX()* en Java) :
    - ***Matrice A*** : objet *Matrice* devant être carrée (*n* équations à *n* inconnues);
    - ***Matrice B*** : objet *Matrice* devant être de dimension [*n*, 1] (ou [1*, n*])
- Admet les opérations suivantes :
    - ***Matrice TrouverXParCramer()*** : retourne une matrice X contenant les valeurs des inconnues en appliquant la règle de Cramer;
        - Avant de procéder, on vérifie d'abord que le déterminant est non nul; s'il est nul, alors il faut afficher un message explicatif à la console et retourner « *null* »;

- ***Matrice TrouverXParInversionMatricielle()*** : retourne une matrice X contenat les valeurs des inconnues en appliquant la méthode d'inversion matricielle;
    - Avant de procéder, on vérifie d'abord que le déterminant est non nul; s'il est nul, alors il faut afficher un message explicatif à la console et retourner « *null* ».
- ***Matrice TrouverXParJacobi(double epsilon)*** : retourne une matrice X contenant les valeurs des inconnues en appliquant la méthode itérative de Jacobi;
    - Avant de procéder, on vérifie d'abord que la condition de convergence est respectée (la dominance diagonale stricte); si on ne peut s'assurer de la convergence, alors on retourne « *null* » après avoir affiché un message explicatif à la console;
    - Le paramètre « epsilon » représente le taux d'écart minimal acceptable entre les valeurs des inconnues de deux itérations successives afin de réussir le test de terminaison, c'est-à-dire de juger la solution comme ayant convergée.

Enfin, ajoutez aussi une méthode permettant d'afficher le système (sous forme d'équations $\boldsymbol{ax_1 + ax_2 + \ldots = b_1}$) à la console ou dans un contrôle utilisateur (p. ex. en redéfinissant la méthode *ToString()* ou une méthode similaire).

CONSEIL : Ne négligez pas d'ajouter tout commentaire pertinent à votre code. Ils ne seront pas évalués comme tel, mais ils pourront tout de même aider lors de la correction à comprendre ce que vous vouliez faire en cas d'erreur.

Bien évidemment, vous aurez besoin d'une classe « ***Main*** » qui permet d'interagir avec ces classes afin de tester vos algorithmes et vérifier leur bon fonctionnement en affichant à la console ou dans des contrôles utilisateurs les matrices/systèmes et les résultats de l'utilisation des méthodes de résolution.

N.B. Vous êtes tenu de faire en sorte que l'utilisateur puisse entrer des matrices ou des systèmes de son choix à l'exécution. De plus, si vous le souhaitez, vous pouvez créer vos instances de classe *Systeme* et *Matrice* directement dans le code de la classe « *Main* » (comme un « *preset* »), mais l'affichage du système/de la matrice et le résultat des opérations doit quand même être fait.

EXIGENCES ET INSTRUCTIONS SUPPLÉMENTAIRES

Langage de programmation

Au niveau du langage de programmation, vous devez utiliser soit l'une ou l'autre des options suivantes :

- C#/VB/C++ avec Microsoft Visual Studio .NET 2017/19;
- Java

Éléments à remettre

Vous devrez remettre l'ensemble des extrants exigés qui seront énumérés ci-dessous dans un seul fichier compressé (en format .zip) dans la section de dépôt des travaux sur le portail du cours **avant le début du cours à la date prévue avant 8h30.**

Dans le cas où vous remettiez le travail en retard, vous ne pourrez alors le déposer sur le portail et vous devrez me le retourner via courriel. Une pénalité de 25% par jour de retard, à compter de l'heure de remise, serait alors appliquée.

Les éléments à remettre sont les suivants :

- Tous les fichiers relatifs au projet de l'application :
    - Si vous travaillez avec Visual Studio .NET, vous devez remettre le répertoire contenant la solution, le ou les projets associés, les fichiers contenant le code source, les exécutables, etc.
    - Si vous travaillez avec Java NetBeans/Eclipse, vous devez remettre le projet NetBeans/Eclipse, ainsi que les fichiers .java et .class, puis le .jar généré.

  Dans tous les cas, assurez-vous que le projet soit prêt à être exécuté directement à l'intérieur de la plateforme de développement, c'est-à-dire sans erreur de compilation et sans configuration spéciale non explicitement donnée. Aucun débogage ne sera fait lors de la correction. C'est TRÈS important. Si je dois reconstruire un nouveau projet et importer les classes, les fichiers de configuration et configurer des paramètres pour vous, il y aura d'importantes pénalités appliquées.

- Un rapport Word ou PDF dans lequel vous devez au minimum indiquer les éléments suivants :
    - Page de présentation;
    - Problèmes et difficultés rencontrés (s'il y a lieu);

- Instructions spéciales d'exécution du programme (s'il y a lieu);
- **Guide d'utilisation avec instructions d'utilisation et impressions d'écran** :
  - Vous devez, à l'intérieur de ce guide, montrez comment les systèmes d'équations sont créés et le résultat de l'appel de chacune des méthodes de résolution des systèmes (les systèmes doivent être affichés d'une quelconque façon à l'écran, ainsi que la méthode utilisée et les valeurs de inconnues) ;
  - Pour un même système, vous devez montrer que le résultat est le même peu importe la méthode;
  - **Montrez également les résultats pour un cas** pour lequel le déterminant de la matrice A est nul et qu'on tente d'utiliser une des deux méthodes directes, ainsi qu'un autre cas où la matrice n'est pas strictement dominante et qu'on tente d'utiliser la méthode de Jacobi.
  - **Montrez un exemple de multiplication de 3 matrices ou plus et le nombre d'opérations de produits pour chacun d'entre eux.**

GRILLE D'ÉVALUATION

Voici les critères d'évaluation et la pondération associée à chacun d'entre eux :

Sujet d'évaluation	***Pts (/10)***
Rapport complet/guide utilisateur (incl. les résultats pour les cas demandés)	1
Classe *Matrice :* - Instanciation/variables d'instance : 0.5 pt - Accès aux/modification des éléments : 0.25 pt - Affichage des éléments : 0.25 pt - Addition matricielle, produit scalaire et matriciel : 0.75 pt - Trace et transposée : 0.5 pt - Déterminant : 1.25 pts - Comatrice : 0.75 pt - Matrice inverse : 0.75 pt	5
Classe *Système :* - Instanciation/variables d'instance : 0.5 pt - Accès aux matrices du système : 0.25 pt - Affichage du système : 0.25 pt - Résolution par règle de Cramer : 0.75 pt - Résolution par inversion matricielle : 0.75 pt - Résolution par la méthode de Jacobi : 1.5 pts	4